**Understanding Recursive Algorithms**

## What is a Recursive Algorithm?

A recursive algorithm is an algorithm that uses recursion to solve a problem. Recursive algorithms typically have two parts:

1. **Base case:**Which is a condition that stops the recursion.
2. **Recursive case:** Which is a call to the function itself with a smaller version of the problem.

## Types of Recursion

There are several different recursion types and terms. These include:

* **Direct recursion:**This is typified by the factorial implementation where the methods call itself.
* **In-Direct recursion:**This happens where one method, say method **A**, calls another method **B**, which then calls method **A**. This involves two or more methods that eventually create a circular call sequence.
* **Head recursion:** The recursive call is made at the beginning of the method.
* **Tail recursion:** The recursive call is the last statement

**How Recursion Simplifies Problems**

Recursion simplifies problems that have a **repetitive or nested structure**, such as:

* Mathematical problems like factorial (n!), Fibonacci series.
* Tree traversal, backtracking (e.g., Sudoku solving), and divide & conquer algorithms (like Merge Sort).
* Financial forecasting, where future values are derived from past values recursively.

**Time Complexity of our code:**

* **Recursive Time Complexity:** O(n) — because each recursive call computes one year.
* **Space Complexity:** O(n) — due to the call stack.

**Optimization Suggestion:**

* Use **memoization** to cache already computed values (though unnecessary here due to linearity).
* For better performance and to avoid stack overflow, we can consider an **iterative approach**

**Optimized Code**:

**public static double futureValueIterative(int years, double rate, double initial) {**

**double result = initial;**

**for (int i = 1; i <= years; i++) {**

**result \*= (1 + rate);**

**}**

**return result;**

**}**